Critical Reading of "Optimization Methods for Logical Inference" [1]

Undergraduate Research Internship

Department of Management Sciences

Winter 2008

Supervisor: Dr. Miguel Anjos

UNIVERSITY OF WATERLOO

Rajesh Kumar Swaminathan

April 12, 2008

Table of Contents

1 Introduction to SAT

The boolean satisfiability (SAT) problem consists of answering the following questions:

1. Is a given proposition (or formula) S satisfiable? That is, is it possible to find truth assignments for the variables (or atomic propositions) in S such that S evaluates to *true*? If the answer to this question is no, we call S *unsatisfiable*.
2. Is a given proposition S a tautology? That is, is every possible combination of 2^n truth assignments a model for S? (A model is any satisfying truth assignment.) This is the same as asking if there does not exist a truth assignment for which S evaluates to *false*. The simplest tautology would be $(x_1 \vee \bar{x}_1)$.
3. Does a proposition S_1 *imply* $(\rightarrow)$ another proposition S_2 defined on the same set of variables? That is, is every model of S_1 a model of S_2 without explicitly solving for all the models of S_1 (if this is even possible)?

One way to answer all three questions is to enumerate all 2^n possible truth assignments, where n is the number of atomic propositions, and to run them through the formula. This number of possible combinations unfortunately grows far too quickly (exponentially) with the number of atomic propositions and becomes unmanageable for even as few as 30 atomic propositions (1 billion possible truth combinations). The problems we are likely targeting have from 1000 to 10,000 atomic propositions in them.

The implication question posed in q. 3 can be rewritten as a satisfiability problem similar to the one posed in q. 1 since asking if $S_1 \rightarrow S_2$ is the same as asking if $(\bar{S}_1 \vee S_2)$ is a tautology. Conversely, every proposition S in a satisfiability problem similar to the one posed in q. 1 can be written as an implication problem like the one posed in q. 3 by simply solving $(T \rightarrow S)$ since this reduces to solving $(F \vee S)$ which is the same as (S). The disjunction of a proposition with another proposition that is always unsatisfiable is pointless and can be simplified by dropping the unsatisfiable proposition. Thus questions 1 and 3 are equivalent and are therefore equally "hard".

Question 2 is a little bit "harder" in an algorithmic sense since one would have to explore a significantly larger feasible set that contains every single possible model which can be as large as 2^n, n being the number of atomic propositions, if the proposition is indeed a tautology.

There is a simple equivalence between all three problems. The following statements are equivalent

1. S_1 implies S_2.
2. $(\bar{S}_1 \vee S_2)$ is a tautology.
3. $(S_1 \wedge \bar{S}_2)$ is unsatisfiable.

since every model of S_1 is also a model of S_2. We showed above that statements 1 and 3 are the same. But questions 2 and 3 are simply negations of each other and are therefore the same. Thus all three statements above are equivalent and a simple equivalence between all three problems has been demonstrated.

2 Special Cases in Propositional Logic

2.1 Basic Concepts

Conjunction in logic is a compound proposition that is true if and only if all of its component propositions are true. Conjunctions are represented by the symbol "$\wedge$" and correspond to a logical AND. Disjunction in logic is a compound proposition that is true if and only if at least one of its

component propositions is true. Disjunctions are represented by the symbol "$\vee$" and correspond to a logical OR. Negations are represented by a "bar" on top of the formula or literal such as $\bar{S}$ or $\bar{x}$ and correspond to a logical NOT.

We therefore have the following simple rules:

1. $(S_1 \wedge S_2)$ is T if and only if both S_1 *and* S_2 are T.
2. $(S_1 \vee S_2)$ is F if and only if both S_1 *and* S_2 are F.
3. S is T if and only if $\bar{S}$ is F.

Basic properties and laws of propositions may be applied without affecting the proposition's models ($\neg S$ is alternate notation for $\bar{S}$):

1. Involutory property of negation: $\neg\neg S = S$
2. De Morgan's Laws:
$$\neg(S_1 \vee S_2) \Leftrightarrow (\bar{S}_1 \wedge \bar{S}_2)$$
$$\neg(S_1 \wedge S_2) \Leftrightarrow (\bar{S}_1 \vee \bar{S}_2)$$
3. Distributive Law: $S_1 \vee (S_2 \wedge S_3) \Leftrightarrow (S_1 \vee S_2) \wedge (S_1 \vee S_3)$

A satisfiability problem posed as a propositional formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of one or more *clauses*, each of which is a disjunction of one or more (possibly negated) literals.

Theorem Any formula in propositional logic is equivalent to a CNF formula whose length is linearly related to the length of the original formula.

That there exist rewriting techniques leading to CNF representations that are polynomially bounded in length was first noted by Tseitin as early as 1968.

Consider the worst-case scenario which is rewriting a formula in *disjunctive normal form* (DNF) as CNF. A DNF formula is a disjunction of *terms*, each of which is a conjunction of literals. An example of a DNF formula is:

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6) \vee (x_7 \wedge x_8)$$

We introduce new propositions x_{12}, x_{34}, x_{56}, x_{78} that represent each conjunction in parenthesis. For each $x_{2j-1,2j}$ $(j = 1, 2, 3, 4)$ we write the clauses $(\bar{x}_{2j-1,2j} \vee x_{2j-1})$ and $(\bar{x}_{2j-1,2j} \vee x_{2j})$

We also need one additional clause to knit the four subformulas together.

$$(x_{12} \vee x_{34} \vee x_{56} \vee x_{78})$$

These clauses put together represent the DNF formula above. It is easy to see that this technique results in a CNF formula with three times the original number of atomic propositions, and twice the original number of clauses plus one additional clause to knit the subformulas together. Both the number of atomic propositions and the number of clauses are therefore linear in growth. □

2.1.1 Unit Resolution

Consider the following example:

$$(x_1) \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5)$$

It is clear that if this formula is to hold true, one would have to set x_1 to 1 since it stands alone as a unit positive clause. We therefore set x_1 to 1 and remove it from the formula. In addition, we also remove all other clauses where x_1 is present as a positive literal since these clauses will be

automatically satisfied. Furthermore, we get rid of all occurrences of $\bar{x}_1$ in any of the remaining clauses since this literal will always be false and a disjunction with something that is always false is pointless.

We are now left with

$$(x_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5)$$

We now repeat the process with $\bar{x}_4$ to obtain the reduced form

$$(x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_5)$$

We now stop as we cannot apply the same process once more due to a lack of unit clauses.

This procedure is referred to as *unit resolution*.

1. If the unit resolution procedure applied to S returns an empty formula then unit resolution has succeeded in finding a satisfying truth assignment for S.
2. On the other hand, if the procedure returns $S = \{\}$, that is the empty clause, then S is unsatisfiable since the only way to obtain an empty clause is to have something of the form $(x_i) \wedge (\bar{x}_i)$ to begin with.

2.2 Integer Linear Programming Models

Integer linear programming deals with linear programming problems where one or more variables are restricted to taking integer values. By introducing suitable bounds on the integer variables (which are expressed as linear inequalities), it is possible to restrict variables to only take values in the nonnegative integers or even just values of 0 or 1. It is the latter "boolean" restriction that captures the semantics of propositional logic since the values of 0 and 1 may be naturally associated with *False* and *True*.

The idea here is to formulate satisfiability of CNF formulas as integer linear programming problems with clauses represented by constraints and atomic propositions represented by 0-1 binary variables.

In general, integer linear programming (ILP) problems may be solved by solving the linear relaxation and then applying a branch-and-bound procedure. But since all of our variables are not only integer but also binary, we may alternatively apply special-case algorithms such as Balas' additive algorithm to solve the binary ILP problem. We may stop these algorithms as soon as a feasible integer solution is found, since we are not optimizing anything here.

Consider, for example, the single clause $x_2 \vee \bar{x}_3 \vee x_4$. This clause may be represented by the inequality $x_2 + (1 - x_3) + x_4 \geq 1$ with x_2, x_3 and x_4 restricted to boolean values of 0 and 1. Thus the SAT problem

$$(x_1) \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5) \tag{1}$$

may be represented by the following set of constraints, one for each clause:

$$\begin{aligned} x_1 &\geq 1 \\ x_2 + (1 - x_3) + x4 &\geq 1 \\ (1 - x_1) + (1 - x_4) &\geq 1 \\ (1 - x_2) + x_3 + x_5 &\geq 1 \\ x_1, \cdots, x_5 &= 0 \text{ or } 1 \end{aligned}$$

Moving all constants to the right hand side, we get the following *clausal* form:

$$\begin{aligned} x_1 &\geq 1 \\ x_2 - x_3 + x_4 &\geq 0 \\ -x_1 - x_4 &\geq -1 \\ -x_2 + x_3 + x_5 &\geq 0 \\ x_1, \cdots, x_5 &= 0 \text{ or } 1 \end{aligned} \tag{2}$$

In general, satisfiability in propositional logic is equivalent to the solvability of the linear system $Ax \geq b, x \in \{0,1\}^n$ where the inequalities $A_i x \geq b_i$ are clausal.

A few quick notes and observations:

1. A is an $m \times n$ matrix of 0s and $\pm$1s, where m is the number of clauses and n is the total number of atomic propositions in the formula. A_{ij} is +1 if x_j is positive in clause i, -1 if negated, and 0 otherwise.
2. $b_i = (1 - \#$ of -1s in row i of matrix $A)$. b can therefore be calculated given a matrix A of 0s and $\pm$1s.
3. The geometric interpretation of a SAT problem reduces to looking for an extreme point of the unit hypercube in $\Re^n$ that is contained in all the half-spaces defined by the clausal inequalities.

We may verify the involutary property of the negation operator "¬" and De Morgan's laws stated previously by looking at their corresponding linear integer constraint representations:

1. $(\neg\neg x_1) \Leftrightarrow 1 - (1 - x_1) \geq 1 \Leftrightarrow x_1 \geq 1 \Leftrightarrow (x_1)$
2. $\neg(x_1 \vee x_2) \Leftrightarrow 1 - (x_1 + x_2) \geq 1 \Leftrightarrow x_1 + x_2 \leq 0$ which is equivalent to the set of constraints

$$\begin{aligned} x_1 &\leq 0 \\ x_2 &\leq 0 \end{aligned}$$

since if any one of x_1 or x_2 (say x_1) were allowed to be strictly positive, it would take the value of 1 (recall the additional constraint $x_i \in \{0,1\}$) and so x_2 would need to be at most -1 to satisfy $x_1 + x_2 \leq 0$ which is of course not possible. We reach a contradiction and hence conclude that both x_1 and x_2 need to be at most 0 if their sum is to be at most 0.

One may now proceed to verify the other half of De Morgan's laws and the distributive property in a similar fashion.

2.2.1 Optimization and Inference

We have already seen that the intersection of clausal half-spaces defines a convex polyhedron. If the additional box constraints $0 \leq x_j \leq 1$ are added, we obtain a *bounded polyhedron* also known as a *polytope*. Satisfiability now essentially implies finding a feasible integer point inside this polytope.

We may check to see if an integer linear programming problem is feasible or not by using a 2-phase method, that is, by adding an *artificial variable* and then trying to optimize the artificial variable to 0. Thus the satisfiability of (1) may be tested by checking the feasibility of (2) which is in turn done by solving the following optimization problem:

$$
\begin{array}{lll}
\text{MIN} & x_0 & \\
\text{s.t.} & x_0 + x_1 & \geq \ 1 \\
 & x_0 + x_2 - x_3 + x_4 & \geq \ 0 \\
 & x_0 - x_1 - x_4 & \geq \ -1 \\
 & x_0 - x_2 + x_3 + x_5 & \geq \ 0
\end{array}
$$

$$x_j \in \{0,1\}, j = 0, 1 \ldots 5$$

The above optimization problem is always feasible with $x_0 = 1$ and all other $x_j = 0$ and an optimal value of $x_0 = 0$. Thus, the original formula is satisfiable if and only if the optimization problem above is solved with x_0 at 0. When $x_0 = 0$, the x_0's in the constraints drop out and we are left with a feasible solution to the original set of constraints put down in (2).

The above optimization problem is called a phase 1 construction which takes the general form

$$
\begin{array}{ll}
\text{MIN} & x_0 \\
\text{s.t.} & x_0 e + Ax \geq b \\
 & x_j \in \{0,1\}, j = 0, 1 \ldots n
\end{array}
$$

where $Ax \geq b$ represents the original clausal inequalities and e is a column of ones.

We therefore now have a way of checking to see if a given proposition is satisfiable or not. How about the inference problems in the form of implications? That is, does a formula S_1 imply another formula S_2 ($S_1 \rightarrow S_2$)? We have already seen that the problem ($S_1 \rightarrow S_2$) is identical to asking if ($\bar{S}_1 \vee S_2$) is a tautology.

To begin with, assume S_1 is CNF and S_2 is given by a single clause C. The optimization model is given by:

$$
\begin{array}{ll}
\text{MIN} & cx \\
\text{s.t.} & Ax \geq b \\
 & x \in \{0,1\}^n
\end{array}
$$

where c is the incidence vector of clause C and $Ax \geq b$ are the clausal inequalities representing S_1. The incidence vector c is constructed by assigning a value of +1 to c_i if the literal x_i is positive in C, -1 if negated, and 0 otherwise.

If this optimization yields a minimum value of 1 - n(C) (1 minus the number of negative literals in C) or larger, S_1 implies S_2. Otherwise the implication does not hold.

Why is this the case?

Suppose that there exists a model that satisfies both S_1 and S_2, then we are guaranteed a feasible solution for the set of constraints

$$
\begin{array}{lll}
Ax & \geq & b \\
cx & \geq & 1 - n(C)
\end{array}
$$

$$x \in \{0,1\}^n$$

Now we take the left-hand-side of the second constraint and move it to the objective function and then try to minimize it. Since $Ax \geq b$ still remains as a constraint (note that we do not associate any artificial variables with this constraint), we are ensured that we are only working with models of S_1 (since $Ax \geq b$ are the clausal inequalities representing S_1).

The question is whether all these models of S_1 are also models of S_2 whose inequality is represented by $cx \geq 1 - n(C)$. If all models of S_1 are also models of S_2, cx would always remain greater than $1-n(C)$ (meaning that $cx \geq 1-n(C)$ always remains feasible when $Ax \geq b$ is) for if there was even one model of S_1 that was not a model of S_2 then the minimization would pick this up to yield an objective value cx that is strictly less than $1 - n(C)$ which causes the inequality $cx \geq 1 - n(C)$ representing S_2 to be violated.

Thus S_1 implies S_2 if and only if the optimal minimum value is greater than or equal to $1-n(C)$.

Now, what if S_2 is given by more than just a clause? The idea is here is to show that $(S_1 \wedge \bar{S}_2)$ is unsatisfiable as discussed in Section 1.

Consider the situation

(i) S_1, CNF, with clausal inequalities $Ax \geq b$

(ii) $\bar{S}_2$, CNF, with clausal inequalities $Bx \geq d$

To test if S_2 is logically implied by S_1 we solve

$$\begin{array}{lll} \text{MIN} & x_0 & \\ \text{s.t.} & x_0 e + Bx \geq d & \text{(corresponds to } \bar{S}_2\text{)} \\ & Ax \geq b & \text{(corresponds to } S_1\text{)} \\ & x \in \{0,1\}^n & \end{array}$$

If this problem is optimized at 0, then both S_1 and $\bar{S}_2$ are true which means there exists a model of S_1 that is not a model of S_2 indicating that S_1 *does not* imply S_2. Thus S_1 implies S_2 if and only if the minimum value of x_0 is 1, for if x_0 were allowed to be 0, the minimization would pick it up. The fact that the minimization does not pick it up indicates that there is no model of S_1 that is not also a model of S_2.

Once again, we do not associate an artificial x_0 with the constraint $Ax \geq b$ since we assume that S_1 is satisfiable. If S_1 had no models to begin with, i.e. if it were unsatisfiable, the implication $S_1 \rightarrow S_2$ always holds.

To conclude, we can always easily rewrite inference problems in propositional logic as integer linear programming (ILP) problems. But in general, ILP problems are just as hard (if not harder in particular instances) to solve as the satisfiability problem itself. It seems then that we have taken a hard problem—the one of satisfiability—and made it even harder by converting it into an ILP problem. However in practice, special mathematical structure makes many ILP problems easy to solve and this happens to be true of many inference problems in propositional logic. We now proceed to investigate special mathematical structure within an ILP representation of SAT by looking at its corresponding linear programming (LP) relaxation.

2.2.2 The Linear Programming Relaxation

The linear programming relaxation consists of taking the ILP problem formulated above and relaxing the integer (binary) constraints $x_j \in \{0,1\}$ to the weaker condition $0 \leq x_j \leq 1$.

The idea is to analyze the properties of this linear programming relaxation to obtain ideas on computational strategies for solving the integer model. It turns out that the linear programming relaxation retains sufficient structure to be a useful representation of the original inference problem. The relaxation provides a means of understanding special structures in propositions that permit efficient solvability of inference.

It is interesting to observe that the actions performed by the unit resolution procedure discussed in Section 2.1.1 earlier are implicitly encoded into the linear inequalities of the linear programming

relaxation. For example, a unit clause (x_j) is given by $x_j \geq 1$. However, there is an explicit upper bound $x_j \leq 1$ on x_j. Thus x_j is fixed to have a value of exactly 1 (as fixed by unit resolution). Similarly, a unit clause with a negated literal $(\bar{x}_k)$ is given by the linear inequality $-x_k \geq 0$, but there exists a lower bound on x_k given by $x_k \geq 0$ which fixes the variable x_k to have a value 0. Also if at any stage of unit resolution two conflicting unit clauses (x_j) and $(\bar{x}_j)$ are obtained, the corresponding implied inequalities are $x_j \geq 1$ and $-x_j \geq 0$, which are in conflict. The linear program is inconsistent and therefore infeasible in such a situation.

Lemma The linear programming relaxation of a unit clause free CNF formula is always feasible with the trivial fractional solution $x_j = \frac{1}{2}$ for all j.

It is easy to convince ourselves that the lemma is indeed true. If the clause contained (two or more) positive-only literals, the left hand side of the corresponding constraint would always add up to 1 or more. For every additional negated literal that is added, the left hand side decreases by a $\frac{1}{2}$, but the right hand side decreases by 1, a larger quantity, thereby preserving the inequality. If the clause contained one posited and one negated literal, the left hand side would sum up to 0 and so would the right hand side.

The lemma implies the following theorem:

Theorem A proposition S has a unit refutation (unsatisfiability of S proved by unit resolution) if and only if the linear programming relaxation of S is infeasible.

For satisfiable propositions, the power of the linear programming relaxation exceeds that of unit resolution. This is because for satisfiable propositions we may get lucky while solving the linear programming relaxation and hit on an integer solution and thus prove satisfiability. The statement implies that there might exist a special class of satisfiable propositions for which unit resolution fails to come up with a solution (perhaps because we're left with a unit clause free formula at some point), but a feasible solution for the linear programming relaxation happens to be integer.

One such class of propositions are called *balanced propositions*. These are propositions whose relaxations are integral polytopes. Using a simplex method to solve the relaxation will guarantee us a vertex feasible solution that is integer and hence prove satisfiability. However, if the proposition itself has no unit clauses, unit resolution would not achieve anything.

For refutable propositions on the other hand, the power of unit resolution and the relaxation are identical. That is, if unit resolution can refute a proposition, then so can solving the linear programming relaxation and vice versa. Inversely, if unit resolution fails to refute a refutable proposition, then the linear programming relaxation also has no hope of refuting it and vice versa.

2.3 Horn Polytopes

We now proceed to analyze special classes of propositions by looking at properties of their linear relaxations and attempting to derive insight into solving their corresponding ILP representations efficiently. One such special class of propositions are Horn propositions.

A *Horn* rule has either no atoms or a single atom in its consequent. A Horn clause must therefore contain *at most* one positive literal since consequents of a rule correspond to positive literals inside a clause. A Horn clause system is a system in which all clauses are Horn.

2.3.1 Horn Resolution

Horn systems are highly structured propositions where satisfiability can be solved in linear time (in the number of literals) using a restricted form of unit resolution. The restricted form of unit

resolution resolves only unit *positive* clauses since a Horn clause system with no unit positive clauses is trivially satisfiable by assigning 0 to the rest of the literals. This is because each clause must then have at least one negated literal, for none of the clauses may contain a single positive literal (it would then be unit positive) or two or more positive literals (it would then be non-Horn). Thus assigning 0 to all literals will ensure all clauses are satisfied simultaneously.

The restricted form of unit resolution can therefore completely solve the satisfiability problem on a Horn system:

1. Look for only unit positive clauses and apply unit resolution on them.
2. If no unit positive clauses are found, assign 0 to all literals in the formula, declare the proposition as satisfiable and stop.
3. If an empty clause is obtained, the proposition is unsatisfiable. Stop.
4. If an empty formula is obtained, unit resolution has succeeded in finding a satisfying truth assignment. Stop.
5. Repeat step 1 using the resulting simplified formula.

In doing so, we take advantage of a basic property that Horn systems are closed under the deletion of literals and clauses and therefore applying unit resolution on a Horn system results in a new system that is also Horn.

Consider the Horn clause system $(x_3) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3)$. Applying unit resolution once yields $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1)$ with x_3 set to 1. In the usual case, we might re-apply unit resolution on the unit clause $(\bar{x}_1)$, but because this system is Horn and there are no other unit positive clauses, we instead assign 0 to both x_1 and x_2 and declare the proposition as satisfiable.

Theorem A *satisfiable* Horn proposition has a *unique minimal model*. A unique minimal model for a satisfiable proposition S is achieved by:

(i) Setting all atoms to T *only* for those atoms that *must* be true in *all* models of S.

(ii) Setting everything else to F.

The restricted form of unit resolution satisfies (i). The minimal model tries to minimize the number of T's assigned to the variables, or equivalently, tries to maximize the number of F's assigned.

The minimal model is unique because of the following argument. Assume there exists two minimal models T_1 and T_2 different from each other. If an atom set to T in T_1 is set to F in T_2 and T_1 is a model, then T_2 cannot be a model because (i) sets *only* those atoms that must be true in all models to T since an atom is set to T only when it shows up as a unit positive clause (positive singleton).

Thus if there exists two minimal models T_1 and T_2, they must be identical to each other. □

Even if there exists more than one unit positive clause to resolve on, the procedure will yield the same minimal model regardless of which clause unit resolution picks to resolve on.

The above theorem also follows from a very strong *closure* property satisfied by the set of models of a Horn proposition.

Lemma If T_1 and T_2 are models for a Horn proposition S, then so is $T_1 \wedge T_2$. (An atom is set to true in $T_1 \wedge T_2$ if and only if it is true in both T_1 and T_2.)

Proof: Assume T_1 and T_2 are models for a Horn proposition S, but $T_1 \wedge T_2$ isn't.

1. If a clause C is negative (meaning no positive literals), then $T_1 \wedge T_2$ must set all literals in C to T since setting even one literal to F will satisfy the clause since all literals are negated. But the only way all literals can be set to T is if both T_1 and T_2 set all corresponding literals to be T which causes both T_1 and T_2 to falsify C since all literals are negated. Therefore

neither T_1 or T_2 can be models and this is a contradiction.

2. If C has a single positive literal x_k then at least one of T_1 or T_2 (say T_1) must set x_k to F, for otherwise $T_1 \wedge T_2$ would satisfy C. But since T_1 is a model, there must be at least one negated literal $\bar{x}_j$ in C (no further positive literals are allowed since C is Horn) set to F, but this would automatically make the corresponding negated literal in $T_1 \wedge T_2$ F causing C to be satisfied. Again a contradiction.

This concludes that models of Horn propositions are closed under the "$\wedge$" operation. □

Definition Two propositions S_1 and S_2 are said to be *equivalent* if they are built on the same ground set of atoms and if they have the same set of models.

It is easy to see that two propositions are equivalent if and only if S_1 implies S_2 *and* S_2 implies S_1. We shall use the notation $S_1 \Leftrightarrow S_2$ for equivalent propositions S_1 and S_2.

Definition A proposition S is called *H-equivalent* (or Horn equivalent) if it is equivalent to some Horn proposition.

So if S is Horn equivalent to some Horn proposition H, the above lemma suggests that the set of models of H and hence of S is closed under the $\wedge$ operation. Now suppose we have a non-Horn proposition S that is closed under the $\wedge$ operation, does it suggest that there exists some Horn proposition H to which S is *H-equivalent*? It turns out that this is indeed the case because of the following reasoning.

If S is non-Horn, it must contain a clause C of the form $x_k \vee x_l \vee D$, where D is a disjunction of zero or more literals. If every model of S satisfies $x_k \vee D$ we can delete x_l from C and obtain an equivalent proposition. Likewise if $x_l \vee D$ is satisfied by all models, we delete x_k from C. We continue this process, gradually deleting the excess positive literals from clauses in S, until we obtain a Horn proposition equivalent to S or we are unable to apply the deletion criteria. If we are unable to apply the deletion criteria it is because we have two models T_1 and T_2 and a clause $x_k \vee x_l \vee D$ such that

$$T_1(x_k) = True,\ T_1(x_l) = T_1(D) = False$$
$$T_2(x_l) = True,\ T_2(x_k) = T_2(D) = False$$

But then $T_1 \wedge T_2$ cannot satisfy C, which contradicts our assumption that the models of S are closed under the $\wedge$ operation. Therefore the deletion process does not get stuck and S is reduced to an equivalent Horn formula.

We thus have the following result:

Theorem The set of models of a proposition is closed under the "$\wedge$" operation if and only if the proposition is *H-equivalent*.

2.3.2 The Integer Least Element of a Horn Polytope

The rich syntactic and semantic structure of Horn propositions is revealed as special integrality properties of the LP relaxation. This helps shape characteristics of the polytopes formed from the linear programming relaxation of Horn propositions, that is, Horn polytopes.

Definition A least element of a polyhedron P is a point $x_{min} \in P$, all of whose individual components are no larger than the corresponding components of any x in P. In mathematical terms, x_{min} is a least element if $x_{min} \leq x\ \forall\ x \in P$.

Of course, not every convex polyhedron has a least element. In the two-dimensional case, the least element, assuming one exists, will be somewhere at the bottom-left of P. If the bottom-most

left-most point is not a vertex of the polyhedron, it is clear that it cannot possibly be a least element. It is also clear that if a least element exists, then it must be unique since if there exists a second least element different from the first, it would automatically imply one of the two least elements is not least anymore.

Theorem A convex polyhedron defined by a system of linear inequalities

$$P = \{ x \mid Ax \geq b, x \geq 0\}$$

has an *integral* least element if the following conditions are met:

1. b must be integral
2. b must be such that P is non-empty
3. each row of A must have at most one positive component
4. all positive components of A must be equal to 1

If $b \leq 0$ it is evident that P contains a least element $x_{min} = 0$. If $b > 0$, [1] (p. 35) proposes a technique to obtain the least element of P using a lower bound escalation scheme which performs a sequence of translations until the least element is resolved.

Since each inequality associated with a Horn clause will have at most one positive coefficient on the left-hand side and further all positive left-hand-side coefficients will be equal to 1, we are assured the existence of an integral least element for all Horn polytopes generated by satisfiable Horn propositions. Further, this least element is the incidence vector of the unique minimum model of the proposition found by Horn resolution since both methods try to maximize the number of F's (0s) assigned to the variables.

It is important to note that although polytopes generated by satisfiable Horn propositions are guaranteed to have an integral least element, the polytope need not itself be integral.

Thus, one can find the integral least element by optimization since we are guaranteed that for satisfiable Horn propositions, a least element exists and this least element needs to be a vertex of the polytope from the way a least element was defined. Hence a simplex algorithm that hops from vertex to vertex will eventually hit upon this integral least element which corresponds to the unique minimal model, thus proving feasibility of the ILP formulation, and hence satisfiability.

Summarizing this idea, for any vector $c \in \Re$, all of whose components are positive (the simplest would be a vector of all ones), the linear program

$$\min \{ c^T x \mid x \in \text{Horn Polytope}\}$$

is optimized uniquely by the integral least element. Of course, this optimization model is nowhere as efficient as Horn resolution in proving satisfiability. However, the *dual* of the optimization model above may provide additional mathematical insight to give us some idea for polytopes generated from *unsatisfiable* Horn propositions.

2.3.3 Dual Integrality of Horn Polytopes

The dual of the integer linear representation of a satisfiability problem has an interesting interpretation discovered by Jeroslow and Wang [2]. When the clauses are unsatisfiable, the values of the dual variables are proportional to the number of times the corresponding clauses serve as premises in a refutation.

Any satisfiability problem can be written as the following 0–1 problem:

$$\begin{array}{rl} \text{MIN} & x_0 \\ \text{s.t.} & x_0 e + Ax \geq a \\ & x_j \in \{0,1\}, j = 0, 1, \ldots, n \end{array}$$

The linear relaxation of this problem is:

$$\begin{array}{rlcll} \text{MIN} & x_0 & & & \\ \text{s.t.} & x_0 e + Ax & \geq & a & (u) \\ & -x & \geq & -e & (v) \\ & x & \geq & 0 & \end{array}$$

and its dual is:

$$\begin{array}{rlcl} \text{MAX} & u^T a - e^T v & & \\ \text{s.t.} & u^T e & \leq & 1 \\ & uA - v & \leq & 0 \\ & -u & \leq & 0 \\ & -v & \leq & 0 \end{array}$$

The dual solution implicitly indicates how many times each clause is used in a resolution proof of unsatisfiability.

Theorem Let $Ax \geq a$ represent an unsatisfiable set of Horn clauses. Then if $(u,\ v)$ is any optimal extreme point solution of the dual, there is an integer N and a refutation proof of unsatisfiability such that Nu_i is the number of times that each clause i is used to obtain the empty clause in the proof.

For example, consider the following *unsatisfiable* Horn clauses:

$$\begin{array}{ccc} x_1 & & \\ \bar{x}_1 & \vee & x_2 \\ & & \bar{x}_2 \end{array}$$

We add an artificial variable x_0 to check satisfiability:

$$\begin{array}{rlcll} \text{MIN} & x_0 & & & \\ \text{s.t.} & x_0 + x_1 & \geq & 1 & (u_1) \\ & x_0 - x_1 + x_2 & \geq & 0 & (u_2) \\ & x_0 - x_2 & \geq & 0 & (u_3) \end{array}$$

The optimal solution to the primal is $\bar{x} = (x_0, x_1, x_2) = (\frac{1}{3}, \frac{2}{3}, \frac{1}{3})$ while the corresponding dual solution is $\bar{u} = (u_1, u_2, u_3) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ which is non-integral. The theorem states that for some N, Nu gives the number of times each clause is used to obtain the empty clause. The refutation is achieved by first resolving the first two clauses to obtain x_2, and then resolving x_2 with the third clause to obtain the empty clause. So each clause is used once and $N = 3$.

Graphically speaking, let us construct a tree where each clause forms a root node. Any two clauses that can be resolved to yield a third resolvent clause is drawn as a child. This process is applied repeatedly until we are left with the null (empty) clause. Once this tree has been constructed, each u_i tells us how many paths there are to get from the i^{th} clause to the null clause.

The u_i's may also be used to perform a simple sensitivity analysis. If a dual solution yields each $u_i > 0$, it means that in at least one proof of infeasibility, every clause is essential. Also, the

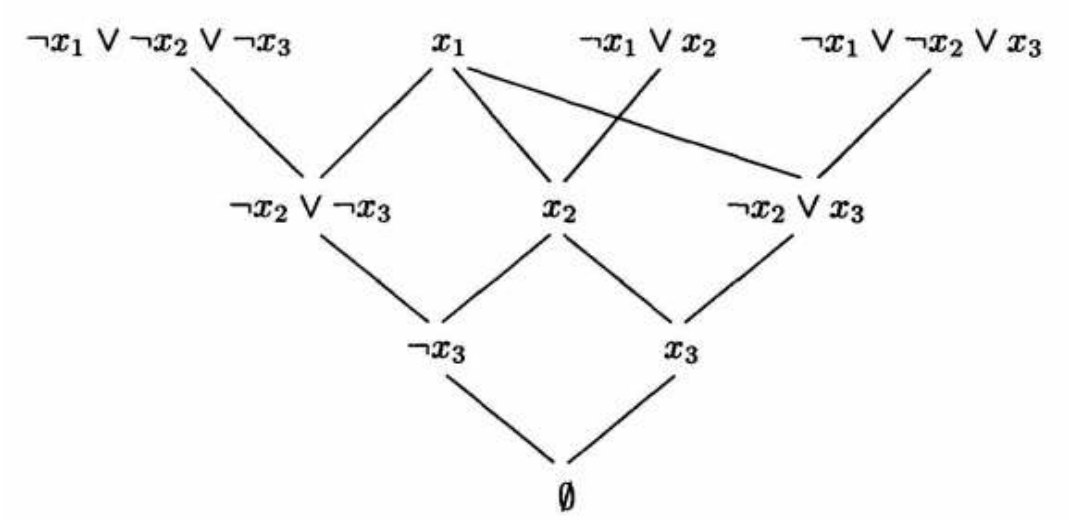

Figure 1: A unit resolution proof of unsatisfiability (taken from [1], p. 40).

clauses i corresponding to larger u_i's might be said to be more "important", in the sense that they are used more often in the proof.

Unfortunately, the dual multipliers u_i do not in general encode the *structure* of a refutation proof and therefore do not represent a complete resolution proof. In other words, they don't tell you anything about *how to* construct the simplest refutation proof. For instance, the dual multipliers $\bar{u} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ obtained above are consistent with two refutations: one in which the first two clauses are resolved first, and one in which the last two clauses are resolved first.

2.4 Quadratic and Renamable Horn Systems

Quadratic propositions allow no more than two literals per clause. This is a simple syntactic restriction.

A proposition is called *disguised* or *renamable Horn* if it can be made Horn by complementing some of the atomic propositions. The problem of recognizing a renamable Horn proposition reduces to the satisfiability of a quadratic formula. This is described in Section 2.4.2. In effect, if we can solve quadratic propositions in linear time, we may be able to solve a new class of non-Horn propositions—propositions that can be made Horn by complementing some of the atoms—in linear time as well.

2.4.1 Satisfiability of Quadratic Systems

As for Horn propositions, there is also a fast linear-time algorithm for checking satisfiability of quadratic propositions. The algorithm uses an underlying graph representation to reveal either a satisfying truth assignment if one exists, or the source of unsatisfiability if not.

A quadratic system is a CNF formula with no more than 2 literals per clause. For example,

$$Q = (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_4) \wedge (x_2) \wedge (\bar{x}_2 \vee \bar{x}_4) \vee (x_3 \vee x_4)$$

is a quadratic formula. Here, we describe a well-known linear-time satisfiability algorithm due to Aspvall, Plass, and Tarjan. The implication graph $G(Q)$ of a quadratic system Q is defined as follows:

1. The vertices of the *directed* graph are the literals.
2. For each clause of the form $(l_i \vee l_j)$ we create two directed edges $(\bar{l}_i, l_j)$ and $(\bar{l}_j, l_i)$ in the graph.
3. For each unit clause of the form (l_k) we create one directed edge $(\bar{l}_k, l_k)$.

An edge of the form (l_i, l_k) has the interpretation that in any satisfying truth assignment, if l_i is set to T, then so is l_j since at an edge of the form (l_i, l_k) must have resulted from a clause of the form $(\bar{l}_i \vee l_k)$ which in order to be satisfied must have l_k necessarily set to T if l_i is T. Therefore for any atomic proposition x_j, if we even have a directed *path* (and not just an edge) from x_j to $\bar{x}_j$ and one from $\bar{x}_j$ to x_j, then Q is not satisfiable.

$G(Q)$ also satisfies a duality property: if we were to reverse the directions of all arrows in the directed graph and complement the names of all vertices, we would get back the same graph. This is because of the way the literals are used to construct the graph. We should get the same graph structure regardless of whether each clause is written as $(l_i \vee l_j)$ or $(l_j \vee l_i)$.

A sufficient and necessary condition for unsatisfiability of Q is the membership of a literal and its complement in what is known as a *strong component* of $G(Q)$. A strong component is a grouping of the vertices of the graph such that within each component, one may get from one vertex to another by strictly moving in the directions of the arrows. All strong components, in topological order, may be identified in linear time by using a depth-first search technique. The topological order simply means that for two strong components S_1 and S_2, if there is an edge from a vertex in S_1 to a vertex in S_2, then S_2 is a successor of S_1, i.e. $S_1 > S_2$. Clearly, there couldn't be an edge back from S_2 to S_1 since if this were the case, both S_1 and S_2 would belong to the same strong component from its definition.

If a literal and its complement existed within the same strong component, it would mean that there is a path comprising of an edge (or a sequence of edges) to get from the literal to its complement and back. However, this would mean that we could satisfy both a literal and its complement simultaneously which is not possible (recall that a connection from l_i to l_j implies that in a satisfying truth assignment, if l_i is set to T, then so is l_j).

The duality property of the graph described above ensures that if S is a strong component of G then so is its dual $\bar{S}$. So if a strong component S satisfies $S = \bar{S}$, then it must contain only complementary pairs of literals. The converse is also true: any literal x_j and its complement $\bar{x}_j$ are both in the same strong component S only if $S = \bar{S}$.

The following procedure describes the linear-time satisfiability algorithm for quadratic systems:

1. Given a quadratic CNF formula Q, construct the implication graph $G(Q)$ according to the rules described above.
2. In *reverse* topological order, process the strong components S of $G(Q)$ as follows. If S is marked true or false, do nothing. If S is unmarked, then if $S = \bar{S}$ stop (Q is unsatisfiable), otherwise mark all literals in S true and all literals in $\bar{S}$ false.

The correctness of the procedure follows from the fact that when the procedure concludes that Q is satisfied, there is no directed path from any vertex marked true to a vertex marked false. This follows from the interpretation of an edge of the form (l_i, l_j), i.e. if l_i is set to T, then so must l_j, meaning that all implications are satisfied.

2.4.2 Recognizing Renamable Horn Systems

As described previously in Section 2.4, a CNF sentence is said to be *renamable* Horn if it can be made Horn by complementing some of its atomic propositions. In order to do this, we need an efficient linear-time algorithm to *identify* a renamable Horn system and disclose atomic propositions that need to be complemented to reduce the system to a Horn formula.

Lewis showed that the disclosure of a renamable Horn formula is reducible to solving the satisfiability of an auxiliary quadratic system. For example, consider the following *non-Horn* proposition:

$$S = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1) \tag{3}$$

Let $u_i \in \{T, F\}$ denote the decision to complement the atomic letter x_i (i.e. $u_i = T$) or leave it alone ($u_i = F$). The conditions on the u_i to ensure that S is renamable Horn are given by the quadratic system

$$Q_S = (u_1 \vee \bar{u}_2) \wedge (u_1 \vee \bar{u}_3) \wedge (\bar{u}_2 \vee \bar{u}_3) \wedge (u_2 \vee u_3)$$

The first clause in Q_S $(u_1 \vee \bar{u}_2)$ states that if x_2 is complemented, then so must x_1 for otherwise the first clause of S would contain two positive literals which is undesirable. In a similar vein, the second clause in Q_S $(u_1 \vee \bar{u}_3)$ states that if x_3 is complemented, then so must x_1 for otherwise the first clause of S would again contain two positive literals which is also undesirable. The third clause of Q_S $(\bar{u}_2 \vee \bar{u}_3)$ states that x_2 and x_3 may not be *both* complemented, while the last clause in Q_S $(u_2 \vee u_3)$ denotes that at least one of either x_2 or x_3 must be complemented in order to make the second clause of S Horn.

We thus conclude that S is renamable Horn if and only if Q_S is satisfiable. The u_i's then tell us which x_i's need to be complemented. In this example, Q_S is satisfiable with $u_1 = u_2 = \mathrm{T}$, $u_3 = \mathrm{F}$ and S is renamed to yield a Horn sentence:

$$S' = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3) \wedge (x_1) \tag{4}$$

Note that unit clauses of S place no restrictions on the u_i's and that Q_S contains one clause for every pair of literals in a non-unit clause of S. A non-unit clause in S containing m literals will therefore generate $\binom{m}{2}$ clauses in Q_S. Thus the length of Q grows quadratically with the length of S. This is a problem.

To rectify this issue, Aspvall [3] used auxiliary variables to obtain a quadratic system *linear* in the size of S. We will describe this technique in detail in the following subsection to see why it works.

2.4.3 Linear-time Recognition of Renamable Horn Systems

We already have a method to generates a set of binary clauses that disclose whether a given system is renamable Horn or not. However, the size of this system of binary clauses grows quadratically with the number of literals in each clause in the original system. In order to avoid this difficulty, Aspvall [3] reformulates the original system using auxiliary (dummy) variables and writes down a secondary system $\tilde{Q}(S)$ whose number of clauses grows linearly in the number of literals of each clause in the original system. Aspvall's reformulation is as follows:

(i) Given any clause $C = (l_1 \vee l_2 \vee \cdots \vee l_k)$ of S with $k \geq 2$, we introduce $(k-1)$ auxiliary variables $w_1^C, w_2^C, \cdots, w_{k-1}^C$.

(ii) The quadratic system corresponding to C is now

$$\begin{aligned}\tilde{Q}_C = \quad & (l_1 \vee w_1^C) \wedge \\ & (\bigwedge_{2 \le t \le k-1} (l_t \vee \bar{w}_{t-1}^C) \wedge (\bar{w}_{t-1}^C \vee w_t^C) \wedge (l_t \vee w_t^C)) \wedge \\ & (l_k \vee \bar{w}_{k-1}^C)\end{aligned}$$

(iii) The final proposition $\tilde{Q}_S$ thus becomes the disjunction of the $\tilde{Q}_C$ of every non-unit clause C:

$$\tilde{Q}_S = \bigwedge_{\substack{C \in S \\ |C| \ge 2}} \tilde{Q}_C$$

The length of $\tilde{Q}_S$ is given as follows: from the definition of $\tilde{Q}_C$ in (ii) we see that the total length of $\tilde{Q}_C$ is $3 \cdot \max\{0, k-2\} \ + \ 2 \le 3|C|$, that is, bounded above by a constant multiple of the length of C. Therefore, $\tilde{Q}_S$ is linear in the size of S, as required. What is left to show is that the solution of the above construction with auxiliary variables is indeed a disclosure of the Horn renamability of S.

Theorem S is renamable Horn if and only if $\tilde{Q}(S)$ is satisfiable. If $\tilde{Q}(S)$ is satisfiable by (u^*, w^*) then a Horn renaming of S is given by the rule for all i that $u_i^* = 1$ means x_i is to be complemented and $u_i^* = 0$ means x_i is not complemented.

Proof:

(i) Let l_i and l_j be *any* two literals appearing in a clause C of S. Then from the definition of $\tilde{Q}_C$ we know that the clauses

$$(l_i \vee w_i^C), (\bar{w}_i^C \vee w_{i+1}^C), \ldots, (\bar{w}_{j-2}^C \vee w_{j-1}^C), (l_j \vee \bar{w}_{j-1}^C)$$

are all contained in $\tilde{Q}_C$. Hence the clause $(l_i \vee l_j)$ is *implied* by the clauses in $\tilde{Q}_C$, meaning that the clause $(l_i \vee l_j)$ can be obtained by resolving all of the above clauses pairwise from left to right. Since the previous method (the method that grew quadratically in length) of disclosing Horn renamability contained one clause for every pair of literals in a non-unit clause of S, we have shown that this new construction involving auxiliary variables can be eventually boiled down to the old construction by resolving appropriate clauses. Hence a satisfying truth assignment (u^*, w^*) for $\tilde{Q}_S$ provides a Horn renaming for S.

For example, consider the simple clause $C = (l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5)$ of S. $\tilde{Q}_C$ for this clause is then given by

$$\begin{array}{ccccccc} & & & & (u_1 \vee w_1) & \wedge \\ (u_2 \vee \bar{w}_1) & \wedge & (\bar{w}_1 \vee w_2) & \wedge & (u_2 \vee w_2) & \wedge \\ (u_3 \vee \bar{w}_2) & \wedge & (\bar{w}_2 \vee w_3) & \wedge & (u_3 \vee w_3) & \wedge \\ (u_4 \vee \bar{w}_3) & \wedge & (\bar{w}_3 \vee w_4) & \wedge & (u_4 \vee w_4) & \wedge \\ (u_5 \vee \bar{w}_4) & & & & & \end{array} \tag{5}$$

The older construction would have $\binom{5}{2} = 10$ binary clauses, one for each pair of literals in C. It is easy to see that all of these 10 clauses are simply an implication of the above $\tilde{Q}_C$

construction containing $(3 \times 3) + 2 = 11$ binary clauses. In other words, all 10 clauses may be obtained by performing repeated resolution on appropriate clauses of $\tilde{Q}_C$. For example, the clause $(u_2 \vee u_4)$ may be obtained by resolving $(u_2 \vee w_2)$ with $(\bar{w}_2 \vee w_3)$ to obtain the resolvent clause $(u_2 \vee w_3)$. This resolvent clause is further resolved with $(u_4 \vee \bar{w}_3)$ to obtain our desired clause $(u_2 \vee u_4)$.

(ii) We now prove the converse of the theorem. Let us begin by assuming S is renamable Horn and that we have a valid renaming given by the u_i's. We need to prove that $\tilde{Q}_S$ will be satisfiable by finding an appropriate assignment for the w_i's.
Consider an arbitrary clause $C = (l_i \vee l_2 \vee \cdots \vee l_k)$ of S. The truth assignment $\bar{u}$ (vector of u_i's) corresponding to the Horn renaming of S must make every u_i $(1 \leq i \leq k)$ true (i.e. rename all l_i's to $\bar{l}_i$) with the possible exception of one literal, say, l_t. This is because once the renaming has been performed, as per the Horn condition, only at most one literal may remain positive. If all l_i's are renamed, we can just assign F to all the w_i's to obtain a model for $\tilde{Q}_S$. It is the other case where l_t isn't renamed that we need to consider.
We now are now faced with the task to figure out what values to assign to the w_i's in order to make $\tilde{Q}_S$ satisfiable when l_t isn't renamed. The required assignment is this: we assign $\bar{w}_i^C = F$ for all $1 \leq i < t$ and $\bar{w}_i^C = T$ for $t \leq i \leq k-1$. The definition of $\tilde{Q}_C$ tell us that the truth assignment (u^*, w^*) is indeed a model for $\tilde{Q}_C$. And since the choice of C was arbitrary, we conclude that the specified (u^*, w^*) makes $\tilde{Q}_S$ satisfiable.
We will illustrate why the above prescription for the assignment of the w_i's work by going back to the simple example introduced in (i). If $C = (l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5)$, then let us suppose all l_i's $(1 \leq i \leq 5)$ except l_3 are renamed. This would make $u_1 = u_2 = u_4 = u_5 = \text{T}$ and $u_3 = \text{F}$. This satisfies 4 out of the 8 clauses in (5). The remaining clauses are the ones with no u_i's in them except for u_3:

$$\begin{array}{ccccccc} & & & & (\bar{w}_1 \vee w_2) & & \\ (u_3 \vee \bar{w}_2) & \wedge & (\bar{w}_2 \vee w_3) & \wedge & (u_3 \vee w_3) & \wedge & \\ & & & & (\bar{w}_3 \vee w_4) & & \end{array}$$

Since u_3 is F, we may remove those literals from the clauses as well:

$$\begin{array}{ccccccc} & & & & (\bar{w}_1 \vee w_2) & & \\ (\bar{w}_2) & \wedge & (\bar{w}_2 \vee w_3) & \wedge & (w_3) & & \\ & & & & (\bar{w}_3 \vee w_4) & & \end{array}$$

It is now easy to see why we assign $\bar{w}_i^C = \text{F}$ for all $1 \leq i < t$ and $\bar{w}_i^C = \text{T}$ for $t \leq i \leq k-1$. Here in this example, we are required to assign $w_2 = \text{F}$ and $w_3 = \text{T}$ since they appear as unit clauses. w_2 is w_{t-1} while w_3 is w_t. Setting $w_2 = \text{F}$ requires us to set w_1 and everything above to F as well because of the clause in the first line $(\bar{w}_1 \vee w_2)$. This means that we have to set all w_i where $1 \leq i < t$ to F. Similarly, setting $w_3 = \text{T}$ requires us to set w_4 to T as well because of the clause in the last row $(\bar{w}_3 \vee w_4)$. Setting w_4 to T will require us to set w_5 (not shown) to T in the clause $(\bar{w}_4 \vee w_5)$ [again not shown] and so on. The effect cascades downwards and requires us to set all w_i where $t \leq i \leq k-1$ to T. $\square$

Let us go back to the S in the example given in (3). The renaming given in (4) gives us $\bar{u} = (u_1, u_2, u_3) = (T, T, F)$. The $\tilde{Q}_S$ corresponding to S is as follows:

$$\begin{array}{lll} & & (u_1 \vee w_1^A) \ \wedge \\ (\bar{u}_2 \vee \bar{w}_1^A) \ \wedge & (\bar{w}_1^A \vee w_2^A) \ \wedge & (\bar{u}_2 \vee w_2^A) \ \wedge \\ (\bar{u}_3 \vee \bar{w}_2^A) & & \\ & \wedge & \\ & & (u_2 \vee w_1^B) \ \wedge \\ (u_3 \vee \bar{w}_1^B) & & \end{array}$$

The first three lines correspond to the $\tilde{Q}_C$ of the first clause of S (clause A) while the last two lines correspond to the $\tilde{Q}_C$ of the second clause of S (clause B).

We may re-write the first clause $(x_1 \vee \bar{x}_2 \vee \bar{x}_3)$ as $(l_1 \vee l_2 \vee l_3)$ by complementing x_2 and x_3. This gives us a different $\bar{u}$. Call this $\bar{u}' = (T, F, T)$. Since the F occurs in the second element in $\bar{u}'$ $(t = 2)$, we set $w_1^A = F$ and $w_2^A = T$. Thus the solution $(u^*, w^{A*}) = ((u_1, u_2, u_3), (w_1^A, w_2^A)) = ((T, T, F), (F, T))$ is the solution obtained by applying the prescription described above. We can check to see that this solution is indeed a model for the $\tilde{Q}_C$ corresponding to the first clause of S.

We can apply the same process for the second clause of S, namely $(x_2 \vee x_3)$. For this clause the renaming is given by $\bar{u} = (u_2, u_3) = (T, F)$. Since all literals in this clause are already posited, we can immediately write down $\bar{w}^B = (w_1^B) = (F)$ since F occurs in the second element of $\bar{u}$ $(t = 2)$. Hence the solution $(u^*, w^{B*}) = ((u_2, u_3), (w_1^B)) = ((T, F), (F))$ is the solution obtained by applying the prescription described above. We can check to see that this solution is indeed a model for the $\tilde{Q}_C$ corresponding to the second clause of S.

Thus S is renamable Horn *if and only if* $\tilde{Q}(S)$ is satisfiable.

2.4.4 Q-Horn Propositions

We have thus far encountered 3 special types of propositions:

1. Horn
2. Renamable Horn
3. Quadratic

all of which admit linear-time satisfiability algorithms. Boros et al. observed that all 3 types of propositions may be unified into a new structure called Q-Horn propositions.

More formally, a Q-Horn proposition is a CNF formula S for which the following linear inequality system is soluble:

$$\sum_{j \,:\, x_j \in P(C_i)} \alpha_j + \sum_{j \,:\, x_j \in N(C_i)} (1 - \alpha_j) \leq 1, \text{ for each clause } C_i$$

$$0 \leq \alpha_j \leq 1, \text{ for all } j$$

where $x_j \in P(C_i)$ refers to the positive literals in C_i and $N(C_i)$ the negative literals.

It is important to note that any fractional feasible solution can be rounded to a half-integer solution such that all $\alpha_j \in \{0, \frac{1}{2}, 1\}$: if $\alpha_j < \frac{1}{2}$ set it to 0, and if $\alpha_j > \frac{1}{2}$ set it to 1. The resulting half-integer solution will still be feasible. Why is this the case?

We may re-write the original linear inequality system as follows (this is the same system with some simple substitutions):

$$\sum_{i\,:\,x_i \in P(C_k)} \alpha_i + \sum_{j\,:\,x_j \in N(C_k)} \alpha_j \leq 1, \text{ for each clause } C_k$$

$$0 \leq \alpha_i, \alpha_j \leq 1, \text{ for all } i, j$$

Clearly at most *two* of $\alpha_i, \alpha_j \geq \frac{1}{2}$ if their sums are to be less than 1. The α's must therefore fall under one of following three cases:

1. Two of the α_i's and α_j's are $\frac{1}{2}$ and all others are 0.
2. Exactly one of the α's $\geq \frac{1}{2}$ and all others are less than $\frac{1}{2}$.
3. All α's are strictly less than half.

In each of the three cases, the rounding gives a feasible $\alpha_i, \alpha_j \in \{0, \frac{1}{2}, 1\}$ solution.

We can check if a given SAT problem is Q-Horn by solving the above linear inequality system and obtaining a feasible fractional solution that can be rounded such that all α_j's are either 0, $\frac{1}{2}$, or 1.

We now show that Horn, renamable Horn, and quadratic propositions are all Q-Horn:

1. *Horn*: Set all $\alpha_j = 1$. Since there is at most one positive literal in each Horn clause, the sum of α's for the positive literals is also at most one.
2. *Renamable Horn*: Set α_j to 0 or 1, depending on the renaming. Set α_j to 0 if x_j is renamed, and 1 otherwise. This works because a clause in a renamable Horn system containing m positive literals (and the rest negative) must have at least $m-1$ literals appearing posited renamed. If exactly $m-1$ literals appearing posited are renamed, then no literals appearing negated can be renamed. But if m literals appearing posited are renamed, at most 1 literal appearing negated can be renamed.
3. *Quadratic*: Set all $\alpha_j = \frac{1}{2}$ since there are at most two literals in every clause.

We can rename all x_j's where $\alpha_j = 0$ to their corresponding complements ($\bar{x}_j$) and set their corresponding α_j's to 1. Thus all Q-Horn formulas have a solution to the above linear inequality system such that $\alpha_j \in \{\frac{1}{2}, 1\}$. Now let X_1 denote the variables that have $\alpha_j = 1$ and X_2 denote the remaining variables where $\alpha_j = \frac{1}{2}$.

We now move the variables (columns) in the incidence matrix around such that the columns of the incidence matrix are partitioned into X_1 ($\alpha_j = 1$) and X_2 ($\alpha_j = \frac{1}{2}$) columns. We may visualize the clauses of the Q-Horn formula to have the following incidence structure now:

$$\begin{pmatrix} H & 0 \\ \Theta & Q \end{pmatrix}$$

where H stands for Horn, Θ for entries that are either 0 or -1, and Q for quadratic.

Once we have arranged the matrix in this structure, we run the following linear-time procedure to obtain a solution:

1. Run Horn resolution on the H clauses.
2. Since Horn resolution fixes the values of the variables (if unsatisfiability hasn't been detected), simplify the remaining clauses using the least model of H.
3. Set all remaining X_1 variables to F since we're guaranteed at least one negated literal in every row of H.

What is left after this procedure is a quadratic formula whose satisfiability can also be solved in linear time as shown in section 2.4.1.

2.4.5 Indexes for CNF Propositions

Based on the linear inequality description of Q-Horn propositions, let us consider an index for any class of CNF propositions that reveals a measure of its difficulty. If S is any SAT instance, let $z(S)$ denote the minimum value attained by the linear program

$$\begin{aligned} \min \quad & z \\ \text{s.t} \quad & \sum_{j \,:\, x_j \in P(C_i)} \alpha_j + \sum_{j \,:\, x_j \in N(C_i)} (1 - \alpha_j) \leq z, \text{ for all } i \\ & 0 \leq \alpha_j \leq 1, \text{ for all } j \end{aligned}$$

The index $z(S)$ sharply delineates the class of "easy" from "hard" satisfiability problems. Any class of propositions in which each proposition S satisfies $z(S) \leq 1 + (c \; log \; n)/n$, where n is the number of atoms in S, admits polynomial time satisfiability algorithms for any $c \in \Re$.

We have shown that Q-Horn formulas S (which are a generalization of Horn, renamable Horn, and quadratic formulas) are exactly those satisfying $z(S) = 1$. Thus for a large n, there is only a narrow range of non-Q-Horn problems that admit polynomial-time algorithms. This is because $z(S)$ is allowed to increase from 1 only by a very small amount given by $(log \; n)/n$ whose value quickly diminishes with increasing n.

Furthermore, a class of propositions in which every proposition S satisfies $z(S) \leq 1 + n^{-\beta}$ is NP-complete for any $\beta < 1$. Since the maximum value of $n^{-\beta}$ is 1, all class of problems with $z(S) \geq 2$ are NP-complete.

It may thus seem that with Q-Horn formulas we may very well be close to the limit of special structure in propositions that admit polynomial-time satisfiability algorithms. This, however, is not the case as we will be shortly seeing.

2.5 Nested Clause Systems

Knuth introduced a new class of propositions called *nested propositions* and provided a linear-time satisfiability algorithm.

Let X be the set of $2n$ possible literals totally ordered by $<$ as follows:

$$x_1 \equiv \bar{x}_1 < x_2 \equiv \bar{x}_2 < \ldots < x_n \equiv \bar{x}_n$$

The total ordering therefore disregards the signs of the literals.

We now make the following definitions and observations to be used later:

1. Given the total ordering, the literals of a clause can be written in increasing order.
2. Clauses, other than unit clauses, have a least literal σ and a greatest literal τ such that $\sigma \neq \tau$. All variables *strictly* between σ and τ are *interior* to that clause.
3. A variable that has not yet appeared interior to any clause is called a *partition* variable and belongs to the partition set. To start with, the partition set is the same as the set of variables (X). As the algorithm processes a new clause, we update the partition set by removing *all* variables that are interior to the new clause whether they appear in the new clause or not.

4. A clause C_i *straddles* C_j if there exists two literals σ and τ in C_1 and a literal α in C_j such that $\sigma < \alpha < \tau$ (note the strict inequalities). For example, using the *natural* total ordering, i.e. $x_1 \equiv \bar{x}_1 < x_2 \equiv \bar{x}_2 < x_3 \equiv \bar{x}_3$ and so on, the clause $(x_1 \vee \bar{x}_3)$ straddles the unit clause (x_2) as well as the binary clause $(\bar{x}_2 \vee \bar{x}_3)$, but *does not* straddle $(\bar{x}_3 \vee x_4)$.
5. Two clauses *overlap* if they straddle each other. An example of overlapping clauses would be $(x_1 \vee x_3)$ and $(x_2 \vee x_4)$.

The definition of a nested system is as follows: A set of clauses is called *nested* if there is *some* ordering of the variables such than *no two clauses overlap.*

2.5.1 Recognition of Nested Propositions

The problem of identifying nestedness in a set of clauses S boils down to planarity-testing of a graph for which well-known linear-time algorithms already exist. We define a bipartite graph G with vertices representing the atomic propositions X $(x_1, x_2, \ldots, x_n)$ on one side and the clauses $c_1, c_2, \ldots, c_m$ on the other. For each literal x_i in each clause c_j, we draw an edge (x_i, c_j). We now extend G by adding a new vertex y and edges (y, x_i) for all i.

If G is planar, then S is nested since planarity implies the clauses do not overlap. The terms "straddle" and "overlap" therefore have graphical interpretations when drawn as a graph, since each literal in each clause is associated with an edge in the graph representation.

2.5.2 Maximum Satisfiability of Nested Clause Systems

Maximum satisfiability (MAX-SAT) subsumes satisfiability (SAT) and tries to maximize the number of clauses satisfied. A proposition is satisfiable if and only if MAX-SAT returns the number of clauses in the system as its output.

MAX-SAT on nested propositions can be solved in linear time. To solve MAX-SAT on nested clauses, we need a valid total order for which the clauses are nested. We assume that the clauses are presented to us in a linear arrangement where each clause appears *after* every clause it straddles. We also assume that the literals of each clause are presented to us in increasing order. This is called a *valid* ordering of the clauses of S. A valid ordering is always possible since there will always be at least one clause that does not straddle any other clause and at least one other clause that is not straddled by any other clause.

A key observation is that once the clauses are presented to us in a valid ordering, then if a variable appears *interior* to a clause C, then it cannot appear in *any* of the subsequent clauses. If it did, then there would either be an overlap or the straddling order (each clause appearing after the clause it straddles) would be violated. As a result, it is enough to remember information regarding the *partition variables* since any future clause *must* have all its variables from this set.

The total number of literals in a set of m nested clauses is therefore at most $2m+n$ (2 non-interior literals per clause and each literal appearing interior to a clause at most once). For convenience, we introduce two dummy variables x_0 and x_{n+1} and a dummy clause $C_{m+1} = (x_0 \vee x_{n+1})$ [define x_0 to be the *least* element in X and x_{n+1} to be the *greatest*].

The MAX-SAT algorithm for nested propositions processes the formula clause by clause (incrementally).

As the algorithm processes each clause incrementally, it can either encounter unit and binary clauses, or it can encounter larger clauses that contain interior variables. In the case of unit or binary clauses, we can just set each variable to first T and then F and pick the assignment that

satisfies the most number of clauses. On the other hand, if the clause *has* interior variables, these variables cannot appear in subsequent clauses since they will be removed from the partition set. This means that a truth assignment to these interior variables will only affect processed clauses. Since this is a MAX-SAT problem, we choose truth-assignments for these variables such that as many processed clauses as possible are satisfied. However, we do not need to check all 2^i truth combinations if there are i interior variables. This is because if z_1, z_2, z_3 are successive variables in the current clause, then z_1 and z_3 could not have occurred simultaneously in any of the previous clauses because if it did, z_2 would have been removed from the partition set (since it is an interior variable) and could not appear in the current clause. This means that assigning T or F to z_1 should not affect any of the clauses in which z_3 occurs although it might affect clauses in which z_2 occur. Thus variables in the current partition interact only with their immediate neighbours and we only have to perform $i + 1$ checks as opposed to 2^i checks.

The complexity of the MAX-SAT algorithm is $O(m+n)$ since we have at most $2m+n$ literals, ignoring the two extra dummy literals we added. The algorithm is therefore linear-time in the sum of the number of clauses and atomic propositions.

The MAX-SAT algorithm for nested propositions is outlined below in pseudo code:

```
1   for i in range(0, n):
2       next[x[i]] = x[i+1]
3   for i in range(0, n):
4       for s in range(0, 1):
5           for l in range(0, 1):
6               maxsat[x[i], s, l] = 0
7   for i in range(1, m+1):
8       least = abs(lit[start[i]])
9       greatest = abs(lit[start[i+1]] - 1)
10      if (least == greatest) || (next[least] == greatest):
11          # update the maxsat array directly
12      else:
13          # compute the new maxsat value and store in newmax
14      next[least] = greatest
15      for s in range(0, 1):
16          for t in range(0, 1):
17              maxsat[least, s, t] = newmax[s, t]
```

3 Summary of Findings

1. We show that the two questions of whether a formula is satisfiable and whether one formula implies another are one and the same. Each form may be converted to the other and therefore both problems are equally hard.
2. In Section 2.2.1 we explain *why* the proposed optimization model for the inference problem of the form $(S_1 \rightarrow S_2)$ works. We begin by assuming that S_1 is CNF and S_2 is given by single clause C. We then extend the argument to the case where S_2 is given by more than just a clause, but is still CNF.
3. For satisfiability problems on Horn systems, we may be able to speed up Horn resolution a little by skipping resolution on a unit positive clause by making the substitution $x_j = \bar{x}_j$ since Horn resolution only resolves unit *positive* clauses. We must of course remember to switch the corresponding value for the propositional atom for each of the modified formula's models. However this substitution is permissible only if *every* other clause containing $\bar{x_j}$ does not already contain a positive clause so as to remain Horn after the substitution.

4 List of Theorems

1. **Theorem** Any formula in propositional logic is equivalent to a CNF formula whose length is linearly related to the length of the original formula.
2. **Theorem** A proposition S has a unit refutation (unsatisfiability of S proved by unit resolution) if and only if the linear programming relaxation of S is infeasible.
3. **Theorem** A *satisfiable* Horn proposition has a *unique minimal model*. A unique minimal model for a satisfiable proposition S is achieved by:
   (i) Setting all atoms to T *only* for those atoms that *must* be true in *all* models of S.
   (ii) Setting everything else to F.
4. **Theorem** The set of models of a proposition is closed under the "$\wedge$" operation if and only if the proposition is *H-equivalent*.
5. **Theorem** A convex polyhedron defined by a system of linear inequalities

$$P = \{\ x \mid Ax \geq b, x \geq 0\}$$

   has an *integral* least element if the following conditions are met:
   (a) b must be integral
   (b) b must be such that P is non-empty
   (c) each row of A must have at most one positive component
   (d) all positive components of A must be equal to 1
6. **Theorem** Let $Ax \geq a$ represent an unsatisfiable set of Horn clauses. Then if $(u,\ v)$ is any optimal extreme point solution of the dual, there is an integer N and a refutation proof of unsatisfiability such that Nu_i is the number of times that each clause i is used to obtain the empty clause in the proof.
   Theorem S is renamable Horn if and only if $\tilde{Q}(S)$ is satisfiable. If $\tilde{Q}(S)$ is satisfiable by (u^*, w^*) then a Horn renaming of S is given by the rule for all i that $u_i^* = 1$ means that x_i is to be complemented and $u_i^* = 0$ means nothing is to be done with x_i.

Bibliography

[1] Vijay Chandru, John N. Hooker, *Optimization Methods for Logical Inference*, John Wiley & Sons, Inc., 1999.

[2] Jeroslow, R. E., and J. Wang, Solving propositional satisfiability problems, *Annals of Mathematics and Artificial Intelligence* **1**, pp. 167-188, 1990.

[3] Aspvall, B., Recognizing disguised NR(1) instances of the satisfiability problem, *Journal of Algorithms* **1**, pp. 97-103, 1980.